Name: Noah Wagner, Dataset: https://www.kaggle.com/datasets/whigmalwhim/steam-releases/

In [13]:
```python
import numpy as np
import pandas as pd

from sklearn.linear_model import LinearRegression
from sklearn.pipeline import Pipeline
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score
```

In [14]:
```python
data = pd.read_csv("game_data_trimmed.csv")
data = data[["release", "peak_players", "total_reviews", "rating", "players_right_now"]]
data.fillna(value=0, inplace = True)

#edit players_right_now column to be numerical (has strings such as "1,234")
data["players_right_now"] = data["players_right_now"].apply(lambda x: int(x.replace(",", "")) if isinstance(x, str) else x)

#modify release dates to be numerical
data["release"] = data["release"].apply(lambda x: int(x.replace("-", "")))

xs = data.drop(columns = ["players_right_now"])
ys = data["players_right_now"]

train_x, test_x, train_y, test_y = train_test_split( xs, ys, train_size = 0.7)
print(train_x, test_x, train_y, test_y)
```

```
       release  peak_players  total_reviews  rating
7858  20220301             4              6   72.17
4427  20221118            39            147   83.60
859   20230217             2             10   75.71
2853  20230215            12             16   57.17
3814  20220711           360            451   87.96
...        ...           ...            ...     ...
3967  20220927         32012          55113   86.64
6844  20220726             4              9   75.00
5419  20221013             8             35   79.23
89    20230331            16             91   87.18
3554  20221222           117           1420   92.31

[7000 rows x 4 columns]         release  peak_players  total_reviews  rating
1350  20230102             1              5   70.84
2093  20230331             3              2   64.08
5521  20220830            71             48   78.75
4734  20220706            62            200   81.90
8314  20220802             2              5   70.84
...        ...           ...            ...     ...
7074  20220910             4              8   74.19
6132  20220118           721           1794   77.00
6190  20211201             3              2   35.92
9057  20220506             4              4   69.20
8532  20220120             3             24   70.68

[3000 rows x 4 columns] 7858       0
4427       0
859        0
2853       0
3814       5
        ...
3967    5934
6844       0
5419       0
89         0
3554       6
Name: players_right_now, Length: 7000, dtype: int64 1350     0
2093     0
5521     4
4734     5
8314     0
        ..
7074     0
6132    18
6190     0
9057     0
8532     0
Name: players_right_now, Length: 3000, dtype: int64
```

In [15]:
```python
steps = [
    ("scale", MinMaxScaler()),
    ("predict", LinearRegression(n_jobs=-1))
]

pipline = Pipeline(steps)

pipline.fit(train_x, train_y)
```

Out[15]:
```
▸ Pipeline
  ▸ MinMaxScaler
  ▸ LinearRegression
```

In [16]:
```python
predict_y = pipline.predict(test_x)
r2_score(test_y, predict_y)
```

Out[16]: 0.4506246964117605

Why I chose the feature columns

The target column describes how many players are currently playing a given game. Below is the rationale for why I included each feature column.

- **release**: Usually older games have fewer players, so I was hoping the model to be able to learn that the lower the date is, the lower the current player could is likely to be.
- **peak_players**: If games have had many players at one point, there is a likely chance that the current amount of players could be some fraction of that. I was expecting the model to learn a positive relationship between this feature and the target.
- **total_reviews**: Peak_players is not enough however. Some games get really popular, but the replayability is low, so player count dies out. This would lead to a low total_review count. I was hoping this feature would help with edge cases where games get really popular, but die out quickly. The more the reviews for a game there are, the more likely it is to still have players.
- **rating**: If a game isn't fun, people are less likely to keep playing. I expected to model to learn a positive coorelation between rating and current player count.

Analyzing performance

After running the model a few times, the prediction r2 score has a very high standard devation. Sometimes it dips down below 0.1, and other times it goes above 0.8. However, I'd say it averages around 0.4. After creating a correlation table, release and rating both have a correlation with players_right_now of less than 0.06. The feature that is the highest coorelated with the target is peak_players (correlation of 0.733), and the total_reviews has a correlation 0.6 (however, it also has a correlation of 0.75 with peak_players). Essentially, the only feature that is usefull to the model is peak_players, so my model can't be that accurate to begin with. Predicting the current players would require more features, as my model has to essentially learn the replayability of a game. To do this, more helpful features would be needed. In addition, I don't think this data is linear, and it would help to run this pipeline through a grid search with transformed regressors. The target also has an overwhelming amount of 0's. The test split could have targets that are all 0, while the train dataset would be getting more of the numerical numbers.

Why I chose r2

Since I chose to do a linear regression model, I thought r2 would be a good way to measure its effectivness. This is because the r2 score measures how well my linear regression model "fits" the data (or how much variance is unaccounted for). Since linear regression is trying to fit the data to a line (or the linear combination of each feature) in order to predict the target, the r2 score measures the inaccuracy of this equation.